# PATENT ABSTRACTS OF JAPAN

(11)Publication number : 11-149385

(43)Date of publication of application : 02.06.1999

(51)Int.Cl. G06F 9/46
G06F 9/46

(21)Application number : 10-008299

(22)Date of filing : 20.01.1998

(71)Applicant : HITACHI LTD

(72)Inventor : SEKIGUCHI TOMONORI
ARAI TOSHIAKI
KANEKO SHIGENORI
ONO HIROSHI
INOUE TARO
SHIBATA TAKASHI

(30)Priority

Priority number : 09248178 Priority date : 12.09.1997 Priority country : JP

(54) MULTI-OS CONSTITUTING METHOD

(57)Abstract:

PROBLEM TO BE SOLVED: To run two OS's on one computer by determining the timing of actuation of an interruption handler according to the execution state of OS's and actuating interruption handlers for the OS's according to the timing.

SOLUTION: A processor 101 when receiving an interruption request from an interruption controller 112 acquires the address of the interruption handler

corresponding to an interruption number from an interruption table that an interruption table register 104 indicates and moves the control to the address to start an interruption process. In this casethe 1st OS reserves computer resources such as a physical memory and an external device that the 2nd OS requires and a management program which is independent of both the OS's takes an external interruption outdetermines which OS's interruption handler should be initiated according to an interruption factorand determines the timing of initiation of the interruption handler according to the execution state of the OS'sthereby initiating the interruption handlers of the respective OS's according to the timing.

## CLAIMS

[Claim(s)]

[Claim 1]He has emulation processing of a privilege instruction which each operating system executes when it has the following and said interruption processing procedure schedules execution of each operating systemand no deviceA multi-OS constitution method operating two or more operating systems simultaneously by one computer.

A procedure which removes hardware resources which other operating systems exploit at the time of starting of the first operating system from an administration object of the first OS about a control method of a computerand is given to other operating systems.

A common interruption processing procedure arranged to a field currently shared between all the operating systems.

[Claim 2]A multi-OS constitution method of claim 1 characterized by comprising the following.

A procedure reserved at the time of initialization processing of the first operating system so that hardware resources for the second operating system cannot be accessed by processing after initialization of the first operating system.

A procedure of loading the second operating system to a reserved physical memoryand starting the second operating system in virtual address space different from the first operating system.
A procedure set up share a part of kernel field of the first operating system with the second operating system.
A procedure of changing a data structure of the first operating system about external interruption which the second operating system receives so that it may not be made to an interrupt inhibit from the first operating system.

[Claim 3]A multi-OS constitution method of claim 1 characterized by comprising the following.
A procedure of determining to which operating system interruption processing is carried out from an interruption factor when interruption is captured.
A procedure of calling a module of the first operating system to the second operating system.
A procedure which returns to the first operating system when processing of the second operating system is completed.

[Claim 4]A multi-OS constitution method of claims 1 thru/or 3 characterized by comprising the following.
A procedure which notifies that to the second operating system when it stops with an obstacle that the first operating system is unrecoverable.
A procedure of permitting interruption of a device which the second operating system has managed at the time of a stop.
A procedure of judging whether the first operating system having stopped at the time of an end of processing of the second operating systeminterrupting and carrying out waiting when having stopped.

[Claim 5]A kernel object file which is a multi-OS constitution method of claims 1 thru/or 4and was confined in a field of exclusive use [ an instruction code and

data which must be shared between both operating systems ] is usedA multi-OS constitution method characterized by the ability to make a shared area small by having the procedure of finding said field of the first kernel object file at the time of the second operating system startingand setting only the field as a shared area.

[Claim 6]Have the followingand it schedules so that the first operating system may be operated by these procedures only at the time of an idol of the second operating systemA multi-OS constitution method shortening response time to interruption which a device which the second operating system manages generates.

A procedure which records it when interruption from a device which is a multi-OS constitution method of claims 1 thru/or 5and the first operating system manages during execution of the second operating system is receivedor a device.

A procedure which starts processing of interruption generated during the second operating system execution when processing of the second operating system is completed and control is returned to the first operating system.

A procedure which changes an operating system immediately and starts interruption processing even if the first operating system is performing interruption from a device which the second operating system has managed.

[Claim 7]A multi-OS constitution method of claims 1 thru/or 5 characterized by comprising the following.

Two or more processors.

A procedure which reserves a processor to the second operating system at the time of initialization of the first operating system by a computer with a device which can specify notifying interruption from an external instrument to a specific processor or a processor group.

A procedure of starting the second operating system by a processor reserved in said procedure.

A procedure set up notify interruption from an external instrument to a processor

in which an operating system which manages each apparatus is runningor a processor group.

[Claim 8]The first operating system comprises two or more files read at the time of starting about a control method of a computerA multi-OS constitution method changing said hardware dependence processing file and realizing claims 1 thru/or 7 when processing of hardware dependence is separated from a file of a kernel main part.
[Claim 9]A multi-OS constitution method of claim 1 characterized by comprising the following.
A procedure reserved at the time of initialization processing of the first operating system so that hardware resources for [ other ] two or more operating systems cannot be accessed by processing after initialization of the first operating system.
A procedure of loading two or more of other operating systems to a reserved physical memoryand starting other operating systems in virtual address space different from the first operating system.
A procedure set up share a part of kernel field of the first operating system with two or more of other operating systems.

[Claim 10]Via a module in a field which is a multi-OS constitution method of claim 9and is shared with two or more operating systems of others of the first operating systemA multi-OS constitution method having the procedure of calling a module of operating systems other than an operating system under execution.
[Claim 11]A multi-OS constitution method of claims 9 thru/or 10 characterized by comprising the following.
A procedure of determining to which operating system of the operating systems currently performed from an interruption factor when interruption is captured interruption processing is carried out. [ two or more ]
A procedure of calling an interruption processing module of an operating system determined by said procedure.

A procedure of returning control to an operating system which was being performed at the time of interruption generating.

[Claim 12]A multi-OS constitution method of claims 9 thru/or 10 characterized by comprising the following.
A procedure which notifies that to two or more of other operating systems which have not stopped when it stops with an obstacle that one or more operating systems are unrecoverable.
A procedure of forbidding interruption which a stopped operating system has managed.
A procedure of forbidding a module call of a stopped operating system.

[Claim 13]Are a multi-OS constitution method of claims 9 thru/or 10and an object file confined in a field of exclusive use [ an instruction code and data which must be shared between two or more operating systems ] is usedBy having the procedure of finding said field of an object file of the first operating system at the time of operating system starting of those other than the first operating systemand setting only the field as a shared area. A multi-OS constitution method being able to make a shared area small.
[Claim 14]A multi-OS constitution method which is a multi-OS constitution method of claims 9 thru/or 10and is characterized by having the procedure of setting up an execution priority among two or more operating systems.
[Claim 15]A multi-OS constitution method of claim 14 characterized by comprising the following.
If a priority of an operating system under execution is lower than a priority of an operating system of a call place when an operating system under execution calls a module of other operating systemsA procedure of changing an execution operating system immediately and carrying out a module call.
If a priority of an operating system under execution is higher than a priority of an operating system of a call placeA procedure of carrying out a module call when

record there being a module call demanda call is postponedprocessing of an operating system whose priority is higher than an operating system of a call place is completed and execution of an operating system of a call place is attained.

[Claim 16]A multi-OS constitution method which is provided with the following and characterized by shortening response time to interruption which a device which an operating system with a high priority manages generates.
A procedure of being a multi-OS constitution method of claim 14recording it and postponing processing until [ when an interrupt which the operating system B of a priority lower than a priority of the operating system A under execution processes occurs ]or a device.
A procedure which starts processing of interruption generated during operating system A execution when processing of the operating system A is completed and the operating system B is performed.
A procedure which interruption from a device which an operating system whose priority is higher than an operating system under execution has managed changes an execution operating system immediatelyand starts interruption processing.

[Claim 17]A multi-OS constitution method of claim 16 characterized by comprising the following.
Interruption which each operating system manages.
A priority currently assigned to each operating system.
A means to set prohibition to permission of interruption from an external instrument according to operating under execution.

[Claim 18]A multi-OS constitution method of claims 9 thru/or 17 characterized by comprising the following.
Two or more processors.

A procedure which reserves a processor to two or more of other operating systems at the time of initialization of the first operating system by a computer with a device which can specify notifying interruption from an external instrument to a specific processor or a processor group.
A procedure of starting two or more of other operating systems by a processor reserved in said procedure.
A procedure set up notify interruption from an external instrument to a processor in which an operating system which manages each apparatus is runningor a processor group.

[Claim 19]About external interruption which is a multi-OS constitution method of claim 9and operating systems other than the first operating system receive. A multi-OS constitution method having the procedure of changing a data structure of the first operating system so that said interruption may not be made to prohibitionfrom the first operating system.
[Claim 20]A multi-OS constitution method which is a multi-OS constitution method of claims 1 thru/or 6 and claims 8 thru/or 17and is characterized by operating two or more operating systems by one processor.

## DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
[Field of the Invention]This invention relates to the multi-OS constitution method which works two or more operating systems on one computer.
[0002]
[Description of the Prior Art]By the usual computerone operating system operatesit manages computer resourcessuch as a processor of a computera memoryand a secondary memoryand the resource schedule is carried out so that

the computer can operate efficiently. There are various kinds of operating systems. It is variousSuch as what is excellent in batch processingwhat is excellent in TSS (TimeSharingSystem)a thing excellent in GUI (Graphical User Interface).

[0003]On the other handthere are needs to perform simultaneously these two or more **** operating system by one computer. For examplein a mainframethere is a demand that the operating system which performs on-line processing accompanying actual businessand the operating system for development want to operate by one computer. Or there is also a demand that the operating system with which GUI is readyand an operating system excellent in real-time requirement want to work simultaneously etc.

[0004]Howevereach operating system assumes managing a computer resource independently.

Coexistence of two or more operating systems is impossible without a certain mechanism.

[0005]As a mechanism in which two or more operating systems are operated on one computerthere is a virtual-machine method (11th volume of OS series VMthe Okazaki ********and KYORITSU SHUPPAN Co.Ltd.) realized with the mainframe. A virtual machine control program occupies and manages all the hardware resourcesvirtualizes itand constitutes a virtual machine from a virtual-machine method. The control section which constitutes a virtual machine virtualizes a physical memoryan input/output-devices deviceexternal interruptionetc.

[0006]For examplethe divided physical memory is served to each virtual machine like the physical memory which begins from the 0th streetand the device number which identifies an input/output device is virtualized similarly. The storage area of the magnetic disk was also divided and it has realized to virtualization of a magnetic disk drive.

[0007]Each operating system is scheduled by a control program so that it may perform on the virtual machine built by the control program. Howeverthe control

section which constitutes a virtual machine for a computer resource from a virtual-machine method in a mainframe thoroughly virtualization and in order to divide is complicatedand a problem.

[0008]When there is no special hardware supportan overhead becomes large and privilege instructionssuch as setting out of the control register which the operating system which operates on a virtual machine publishesand input/output instructionare problemsin order to have to emulate with a virtual machine control program. With the mainframe which mounts the virtual machinehardwaressuch as a processor function special to virtual machines and a microcodeare addedand reduction of overheads is actually aimed at. Since the virtual-machine method aims at virtualizing a computer resource thoroughlyit is complicatedfor highly-efficient-izing of a virtual machinea special hardware mechanism is still more nearly required for itand it is a problem.

[0009]On the other handthere is a microkernel as art of providing the interface of two or more operating systems by one computer. In a microkernelthe operating system server which provides on a microkernel the operating system function shown to a user is builtand a user exploits a computer resource via the server. A user can be provided with various operating system environment if the server for every operating system is prepared.

[0010]Howeverit is necessary to build an operating system server newly in a microkernel method according to a microkernel. In many casesa certain operating system will be changed now so that it may operate on a microkernelbut. The central portion of kernelssuch as scheduling and memory managementwill also be changedthere are many changed partsand in order that a changed part may attain to the central portion of an operating systemalteration work is complicatedis not easy and is a problem.

[0011]Although an operating system server will use service of a microkernelthis is not being the usual operating systemserves as an overhead and brings about degradation.

[0012]

[Problem(s) to be Solved by the Invention]The conventional virtual-machine method was based on the technique of virtualizing total-session machine resources in order to operate two or more operating systems simultaneously. Howeverby this methodthere is a problem to which a control program becomes complicated. Since the emulation of a privilege instruction is needed by this methodhardware special to obtaining performance is requiredand a problem.
[0013]This invention realizes the concurrency of two or more operating systems without special hardware by change of the initialization processing portion of an operating systemand an interruption custodial areaand the addition of an interruption control program. In this inventionsince the emulation of a privilege instruction is unnecessarya new overhead is not followed on execution of each operating system.
[0014]According to this inventionthe function which complements the first operating system can be added easilyand construction of a highly efficient computer system is attained. Since the function to completely operate independently with the first operating system is incorporable unlike a device driverit also becomes possible to add the high reliability-ized function independent of the first operating system.
[0015]In the method of constituting two or more multi operating system environment from a microkernel methodthere is a problem that construction of the operating system server which provides the interface of each operating system is difficult. According to this inventionsince the change to an operating system interrupts with an initialization portion and is limited only to a custodial areait can constitute multi operating system environment easily.
[0016]
[Means for Solving the Problem]This invention reserves computer resources which the second OS needssuch as a physical memory and an outside devicein the first OSand a control program which became independent of both of the OS's seizes external interruptionTwo OS's are operated by one computer of which OS an interrupt handler should be started according to an interruption factorand by

determiningdetermining timing which starts an interrupt handler according to a run state of OSbasing itand starting an interrupt handler of each OS.

[0017]

[Embodiment of the Invention]An embodiment of the invention is described.

[0018]Hereafteran embodiment of the invention is described using a drawing. Drawing 1 is a figure showing the composition of the computer 100 in an embodiment of the invention.

[0019]The computer 100 comprises the memory storage 118 which equips [ processor 101main memory unit 102bus 109interrupt signal line 110clock interruption generation machine 111and interrupt control ] 112and stores the boot procedureand the interruption bus 119.

[0020]The interrupt signal line 110 has connected external input/output devices and the interrupt control device 112. If an external instrument generates interruptionvia the interrupt signal line 110a signal is receivedand the interrupt control device 112 will evaluate this signaland will pass the interrupt control device 112 to the processor 101 via the interruption bus 119.

[0021]The clock interruption generation machine 111 generates periodic interruption.

[0022]The interrupt control device 112 receives the interrupt request from an external instrumentgenerates the interrupt signal evaluated according to the requiring agencyand sends it to the processor 101. Suppose that it can avoid notifying the interrupt signal from specific apparatus to the processor 101 with the directions from the processor 101.

[0023]The processor 101 comprises the arithmetic unit 103the interruption table register 104the page table register 105and the address conversion device 106.

[0024]The interruption table register 104 is pointing to the virtual address of the interruption table 107. Although the details of an interruption table are mentioned laterthe start address of the interrupt handler for every interrupt number is recorded. By drawing 1it interrupted with the interruption table register 104and the dashed line has indicated connection of the table 107 in order for the

interruption table register 104 to interrupt and to point to the virtual address of a table. If an interrupt occursthe processor 101 will receive the interrupt number evaluated from the interrupt control device 112. It sinks below this number as an indexan interrupt handler address is acquired from the table 107and control is passed to an interrupt handler.

[0025]The page table register 105 is pointing to the page table 108. The page table register 105 stores the physical address of the page table 108.

[0026]The address conversion device 106 receives the instruction address which an arithmetic unit requiresor the address with which the operand is storedand carries out virtual-real address conversion based on the contents of the page table 108 which the page table register 105 has pointed out.

[0027]The keyboard 113the display 114the magnetic disk 115and the other external instruments 116 and 117 have connected with the computer 100 as an external I/O device. The apparatus except the display 114 is connected to the interrupt control device 112 by the interrupt signal line 110.

[0028]The contents of the main memory unit 102 are explained briefly. Nowtwo operating systems are operating by the computer 101. Each will be called the first operating system and the second operating system. If a computer is startedit will be set up so that the first OS may startand the external instruments 116 and 117 presuppose that it is apparatus managed by the second OS.

[0029]The first OS is the initialization processing at the time of startingand reserves a physical memory field for second OS in this case to other operating systems. That isthe first OS secures a physical memory field so that the physical memory field reserved for second OS cannot be used. Drawing 1 shows signs that the second OS is loaded to this taken field.

[0030]In the initialization process of the first OSthe interrupt number and I/O address which the external instruments 116 and 117 use from the first OS are reserved noting that it is already ending with use.

[0031]The first OS has a common area which can be referred to from all the operating systems. The interface module etc. which can be called are stored in

the common area from the interruption table 107an interruption control programan interrupt handlerand each operating system.

[0032]The operation outline of an embodiment of the invention is explained. According to this embodimentpriority is given to the second OS over the first OSand it operates. Giving priorityoperating shows that the first OS can operate only when the second OS is an idle state. Unless processing of the second OS is completedthe first OS cannot operate.

[0033]If the device which the second OS manages generates interruptionprocessing of the first OS will be interrupted and it will move from control to the second OS. Even if the first interrupt occurs during second OS executionthe interruption processing is postponed until the first OS is performed.

[0034]The first OS and second OS are classified clearly and it prevents from accessing them mutually except the common area which arranges an interrupt handler etc. It has prevented two operating systems' accessing a mutual field accidentallyand an obstacle occurring by this.

[0035]Hereafterthe embodiment of this invention which realizes the above-mentioned function is described.

[0036]Drawing 2 is a figure showing notionally the relation of two operating systems in an embodiment of the invention. Each operating system holds the address space which became independentrespectively. 201 is the virtual space of the first OS and 202 shows the virtual space of the second OS. Herethe real storage corresponding to the space 202 of the second OS becomes a field of the second OS of the main memory 102 of drawing 1.

[0037]The map of the common area 203 is carried out to a part of virtual space. The real storage corresponding to the common area 203 is the field shown as a common area of the main memory 102 of drawing 1. The common area 203 is a part of field of the kernel of the first OS from the first. When the procedure which loads the second OS builds the address space 202it creates the page table for the second OS so that the common area 203 may be mapped in the address space 202. This procedure is mentioned later.

[0038]Drawing 2 shows the hardware which each operating system manages. It is shown that the first OS manages the keyboard 113the display 114and the magnetic disk 115and the second OS manages the input/output devices 116 and 117. Although the clock 111 and the interrupt control device 112 are hardwares which the first OS has managed from the firstthey show that the program in the common area 203 manages.

[0039]Nextthe composition of a page table is explained. Drawing 3 shows the composition of the page table in the embodiment of operation of this invention.

[0040]300 is a page table. The page table 300 has an entry which describes each virtual page for every virtual page of the virtual address space of the processor 101. Each entry is constituted by the effective bits 301 and the physical page number 302.

[0041]The effective bits 301 show whether the physical page corresponding to the virtual page is assignedor it is got blocked and virtual-real address conversion is possible. For examplesince effective bits are not setthe virtual page 3 of the page table 300 shows that the physical page corresponding to the virtual page 3 does not exist. If access to the virtual page to which the effective bits 301 are not set occursa processor will generate a page fault.

[0042]The physical page number 302 is recording the physical page number corresponding to a virtual page.

[0043]The address conversion device 106 changes into a real address the virtual address which the arithmetic unit 103 generates with reference to the contents of the page table indicating the page table register 105. Refer to the main memory unit 102 for the processor 101 with the real address obtained by conversion.

[0044]The space which became independent by changing a page table can be builtand construction of the space of the first operating system shown in drawing 2 and the space of the second operating system is possible. About the common area 203if it sets up carry out the map of the same physical page to the portion corresponding to the common area of the page table of both operating systemsa common area is realizable.

[0045]Nextthe composition of an interruption table is explained. Drawing 4 shows the composition of the interruption table.
[0046]400 interrupts and it is a table. The interruption table 400 is recording the virtual address 401 of an interrupt handler for every interrupt number which the processor 101 receives from the interrupt control device 112. If an interrupt request is received from the interrupt control device 112the processor 101The address of the interrupt handler corresponding to an interrupt number is acquired from the interruption table 400 to which the interruption table register 104 is pointingand interruption processing is started by moving control to the address.
[0047]Drawing 5 shows the interrupt control device 112. The interrupt control device 112 has the interruption mask register 501 and the selecting arrangement 502.
[0048]The input/output device which generates interruption is connected with the interrupt control device 112 by the interrupt signal line 110. A priority is attached by whether interruption which input/output devices generate is connected to the signal wire of interrupt signal line 110 throat. Herethe interrupt signal corresponding to the interruption No. 0 presupposes that it is interruption with the highest priority.
[0049]The interrupt signal line 110 is connected to the selecting arrangement 502. If an interrupt signal is receivedthe selecting arrangement 502 will record that there is unsettled interruption until it reports that the processor received the interruption.
[0050]The interruption mask register 501 is recording whether I may notify to a processor interruption which input/output devices generated. The interruption mask register 501 can be set up by input/output instruction from the processor 101.
[0051]When the selecting arrangement 502 receives an interrupt request from the interrupt signal line 110When the contents of the interruption mask register 501 are rewrittenit decides whether compare the contents of the interruption mask register 502 with unsettled interruption which the selecting arrangement

502 is recordingand notify interruption to a processor. It is set up that interruption to the interruption mask register 501 is specifically possible among unsettled interruption which the selecting arrangement 502 is recordingand it notifies to a processor sequentially from the highest interruption of a priority. About selected interruptionthe selecting arrangement 502 sends the number signal corresponding to the interrupt signal to notify to the processor 101 by interruption bus 119 course.

[0052]The processor 101 can cancel the unsettled interruption record currently recorded on the selecting arrangement 502 by input/output instructionwhen interruption is received.

[0053]Nextthe boot procedure of a computer in an embodiment of the invention is explained.

[0054]The portion which a boot procedure begins is stored in 118 which is the memory storage only for reading. The memory storage 118 is connected to the processor 101 via the bus 109 so that a map may be carried out to the decided address with the physical address space of a processor. This procedure carries out detection of hardware constitutionsand loading to the main memory of the program which loads an operating system kernel.

[0055]If the processor 101 is resetthe processor 101 will move control to the physical address defined beforehand. The memory storage 118 stores the program executed at this timeand the map is carried out to the physical address space so that control can be passed to this programwhen the processor 101 is reset.

[0056]The program stored in the memory storage 118 loads the kernel loader of the first OS stored in the magnetic disk drive 112 to the main memory unit 102and performs it. The program which a kernel loader has in the position as which the magnetic disk drive 112 was determined beforehandand was stored in the memory storage 118 can find this easily.

[0057]The procedure of a kernel loader is explained. Drawing 6 is a flow chart in an embodiment of the invention which shows the procedure of the kernel loader

of an operating system.

[0058]This kernel loader understands the file system organization of an operating systempinpoints the storing position of a file from a file nameand it is constituted so that it can read into main memory.

[0059]The procedure of a kernel loader is explained. Firstthe main memory list 1101the load module list 1104and the device list 1102 which are passed to a kernel as a parameter are initializedand the page table field for kernels is assigned (Step 601). The composition of three lists is mentioned later.

[0060]The main memory list 1101 is a data structure which shows the using state of the main memory unit 102and when a kernel loader assigns a physical memory by subsequent processingsit refers to and changes and it carries out the main memory list 1101.

[0061]Nextinspection (Step 602) of hardware constitutions and creation (Step 603) of hardware-constitutions data are carried out. in Step 602 -- the computer 100 -- what kind of -- it is connected or hardware inspects. In continuing Step 603the device list 1102 which is a data structure about hardware constitutions is created based on the result of Step 602. An operating system kernel carries out kernel initialization processing with reference to this device list 1102.

[0062]Nextthe configuration information 700 on an operating system kernel is read from the magnetic disk drive 112and the address of configuration information is set as the parameter table 1100 (Step 604). The kernel of the operating system may comprise two or more filesas it was called the file of a kernel main partand the file of other device drivers. The configuration information 700 is stored in the magnetic disk 112 by the file name decided beforehandand the load program can find this.

[0063]The data structure of the kernel configuration information in the embodiment of this invention is shown in drawing 7. 700 shows the contents of the file which is recording kernel configuration information. The configuration information file 700 stores the data which a kernel loader and an operating system refer to. The data stored is named and the program can acquire the data

corresponding to it from a name. In the example of drawing 7there is an entry called an object file (701) in a nameand the data is stored in 702. It is assumed that the data (704) for the second OS is stored at secondary OS.

[0064]Explanation of the procedure of a kernel loader is continued. After reading the configuration information 700about all the kernel configuration files stored in the data in which the name of the object file in the configuration information 700 was given. It reads into the main memory unit 102 (Step 606)an entry is added to the load module list 1104 (Step 607)and the page table for kernels is set up (Step 608). Herethe object file of a file name called kerneldriver1and driver2 is loaded.

[0065]Addition of a load module list entry and setting out of the page table for kernels are carried out according to the data stored in the object file loaded to the main memory 102. The virtual address which carries out the map of the file contentthe size of a fileetc. are contained in the object file which constitutes a kernel. A page table is built with reference to this. The data structure of an object file is mentioned later.

[0066]Finally the page table register 105 is set as the address of the built page tableA processor is made to shift to virtual address translator mode (Step 609)Control is passed to the initialization routine of a kernel by making into a parameter the main memory list 1101the device list 1102 and the kernel configuration information table 1103 which were builtand the parameter table 1110 which comprises the group of the load object list 1104 (Step 610). The entry point of the kernel is recorded on the data in a kernel file.

[0067]Nextthe structure of the object file which constitutes a kernel is explained. Drawing 8 is a figure showing the structure of the object file which constitutes the kernel in an embodiment of the invention.

[0068]800 shows the whole object file. The object file 800 comprises a header part of 801 thru/or 811and section portions of 812 thru/or 813.

[0069]The composition of a header part is explained. The header map address 801 and the header size 802 have described the storing position in the kernel space of object file 800 header part. A header part is read into the address

currently recorded on the header map address 801.
[0070]The initialization entry 803 is recording the address of the routine for initialization of the object file 800. A kernel finds initialization routine with reference to the initialization entry 803 of each object filewhen calling the initialization routine of each object file at the time of kernel initialization.
[0071]804 sections are recording the number of the sections included in the object file 800. A section is a continuous data area in an object fileand mapping to the virtual space of a kernel is determined by making this into a unit. For examplethe object file includes the section where the execution code is storedand the section which stores the data which the object file refers to. These sections are created by a compiler at the time of object file creation.
[0072]The external reference table offset 805 and the external reference table size 806 have described the external reference tables 810 which store the information on the public presentation reference of other object files which the execution code in this object file refers to. The external reference tables 810 are contained in the header part of the object file 800and the external reference table offset 805 is recording offset of the external reference tables 810 from the head of a header.
[0073]The public presentation reference table offset 807 and the public presentation reference table size 808 have described the public presentation reference table 811 which stores the information on the module and data which this object file opens to the execution code of other object files. The public presentation reference table 811 is included in the header part of the object file 800and the public presentation reference table offset 807 is recording offset of the public presentation reference table 811 from the head of a header.
[0074]The section data 809 stores the data about each section included in the object file 800. Section data has only a number with 804 sections. The composition of section data is mentioned later.
[0075]The external reference tables 810 and the public presentation reference table 811 constitute a header part following section data.

[0076]After the header partthe main parts 812 and 813 of each section are stored.
[0077]The composition of the section data 809 is explained. The section start offset 820 and the section size 821 are recording start offset of the section concerned within the object file 800and the size of the section concerned.
[0078]The map of the section is carried out to the virtual space of a kernel so that it may be arranged to the address recorded on the section map address 822. The character string which shows the name of the section concerned is stored in the section name 823.
[0079]The structure of external reference tables is explained. Drawing 9 shows the structure of external reference tables. Several 901 of the external reference information included on this table is stored in the head of the table 810.
[0080]Thenthe object file name 902 and the external reference name 903 are stored. The object file name 902 and the external reference name 903 store the offset value to the character string table 905and the name by a actual character string is stored in the character string table 905.
[0081]The actual address of the external reference described by this external reference entry concerned is stored in the external reference address 904. When an object file is loaded to main memorya kernel acquires a function or the address of data with reference to the table of public presentation reference of an object file including the external reference of this **and sets it as the external reference address 904. With reference to the address stored in the external-function address 904the execution code of the object file is compiled so that the function designator in other object files and reference of data may be carried outand execution of the function in other object modules and data reference are possible.
[0082]The object file name 902the external reference name 903and the external reference address 904 define one external referenceand these entries are arranged continuously only the number currently recorded on 901 external reference. After thatthe character string table 905 is stored. The character string table stores the character string of an object file name and an external reference

name.

[0083]The structure of a public presentation reference table is explained. Drawing 10 is a figure showing the structure of a public presentation reference table.

[0084]Several 1001 of the reference name opened to other object modules by this public presentation reference table 811 is recorded on the head of the table 811. One public presentation reference is described by the open reference name 1002 and the open reference address 1003. The open reference name 1002 stores the offset value to the character string table 1004and the name by a actual character string is stored in the character string table 1004. The open reference address 1003 stores the address corresponding to this reference.

[0085]Nextthe composition of the hardware-constitutions data which the boot procedure which begins from Step 601 createsand load object data is explained. Drawing 11 is a figure showing the composition of hardware-constitutions data and load object data.

[0086]The parameter table 1100 is a data structure which a kernel loader creates. Since three lists which begin from the parameter table 1100 are arranged in the virtual space of the kernel which a loader buildsthey can be referred to from a kernel.

[0087]The parameter table 1100 holds the pointer to the head of three lists which the loader builtand the pointer to one table. Three lists are the main memory list 1101the device list 1102and the load object list 1104and one table is the kernel configuration information table 1103. Each is explained.

[0088]The main memory list 1101 is a list of the main memory block descriptive data 1110. The main memory block descriptive data 1110 comprise the pointer 1114 to the base address 1111the block size 1112the block using state 1113and the following main memory block descriptive data.

[0089]Main memory block descriptive data are recording the using state about a certain continuous main storage area. The base address 1111 shows the start physical address of a continuation fieldand the block size 1112 stores

continuation area size. The value which assigns with whether the continuation field concerned of the block using state 1113 is intact and a loaderand shows whether it is ending is stored. And the pointer 1114 to the following entry constitutes the list. In drawing 11the next entry of 1110 is 1120. The utilizing state of a physical memory can be known by referring to the main memory list 1101.

[0090]The device list 1102 stores the data about the hardware device which the kernel loader detectedand is created at Step 603. The device list 1103 is a list which consists of the device data 1150. The device data 1150 comprises the pointer 1153 to the device type 1151the device information 1152and the following device data.

[0091]The value which shows the kind of device with which the device type 1151 is described by this device data entry 1150 is stored. The device information 1152 stores data peculiar to the kind of device. For examplean interrupt numberan I/O Addressetc. are equivalent to it. And the pointer 1153 to the following entry constitutes the list.

[0092]The pointer 1103 to the kernel configuration information table is pointing to the contents of the kernel configuration information file 700 which the kernel loader read into the main memory 102.

[0093]The load object list 1104 holds the data about the object file which the kernel loader loaded to main memory. A load object list is a list of the load object data 1130. The load object data 1130 comprises the pointer 1133 to the object file name 1131the object address 1132and the following load object data.

[0094]The object file name 1131 is a file name of the object file described with the load object data 1130. The object address 1132 stores the address of the kernel space where the header area of the object file concerned is loaded. And the pointer 1133 to the following entry constitutes the list.

[0095]The load object list 1104 is simultaneously createdwhen a kernel loader reads the object file which constitutes a kernel (Step 607).

[0096]Nextthe initialization procedure of the first OS in an embodiment of the

invention is explained. Drawing 12 is a flow chart which shows the initialization procedure of the first OS.

[0097]Firstwith reference to the load object list 1104 in the parameter table 1100 passed as a parameterexternal reference address solution of the object file which the kernel loader loaded is carried out (Step 1201). In address solutionthe external reference address 904 of the external reference tables 810 in each object file is determined. An address is determined with reference to the public presentation reference table 811 of each object file.

[0098]A main storage area is secured to second OS with reference to the continuing main memory list 1101 of parameter tables 1100 passed as a parameter at the time of kernel starting in Step 1202.

[0099]Specificallythe information on the second OS is taken out from the kernel configuration information table 700. The configuration information on the second OS is stored in 704 in the example of drawing 7. The size of the main memory which should be secured is determined with reference to this configuration information 704. And the contents of the empty block entry of the main memory list 1101 are changedand a main storage area is assigned. This processing is carried outbefore the first OS is vacant and beginning memory management.

[0100]Therebyif it sees from the first OSthe main storage area assigned to the second OS will not existand being referred to of it from the first OS will be lost. Thereforethe assigned field turns into a main storage area which the second OS can use freely. This is equivalent to the field of the second OS of drawing 1.

[0101]The data structure inside a kernel is initialized in the following step 1203. Initialization of the device management table described later is also included in this initialization.

[0102]The device which the second OS manages is reserved in Step 1204. Reserving here is preventing from using from the first OS. Specificallyregistration to the device management table which the first OS has managed is carried out.

[0103]The device resources which the second OS manages are decided with reference to the configuration information on the second OS stored in the table

700 which the kernel configuration information table 1103 of the parameter table 1100 points out. According to this embodimentthe data stored in 704 of drawing 7 is equivalent to it.

[0104]A device management table is explained. Drawing 13 is a figure showing the structure of the device management table of the first OS. A device management table turns into the interrupt vector table management table 1300 from two data structures of the I/O Address management list 1310.

[0105]The interrupt vector table management table 1300 stores the value which shows whether the first OS uses the interrupt number about each interrupt number which the processor 101 receives. A kernel gives the right to use whether this table 1300 is inspected and the demanded interrupt number is used and the interrupt number which inspectedand was demanded only when that was not right to a device driverwhen a device driver requires an interrupt number at the time of initialization. When describing that it is already ending with usethe device can be used from the first OS.

[0106]The input/output devices 116 and 117 of drawing 2 are explained as an example. It is assumed that the input/output devices 116 and 117 are required as the interrupt numbers 4 and 5respectively. The input/output devices 116 and 117 are devices which the second OS manages. The interrupt number which the input/output devices 116 and 117 require is recorded on the configuration information 704 on the second OS of the kernel configuration information table 700. In Step 1204the value which shows using for the entry of the interrupt numbers 4 and 5 of an interrupt vector table management table is stored with reference to this configuration information 704. Since this processing is carried out before the first OS starts device managementit becomes impossible for the first OS to access the input/output devices 116 and 117andas for Lycium chinensethe devices 116 and 117 are made [ the OS ] under management of the second OS.

[0107]The same may be said of the I/O Address management list 1310. The I/O Address management list 1310 is a list which consists of the entry 1320

expressing the I/O Address range. The entry 1320 serves as the I/O Address range 1321 which the first OS uses from the pointer 1322 to the next entry for constituting a list. When a device driver requires the I/O Address range at the time of initializationlike the interrupt vector table management table 1300 a kernelThat address range is already used or it inspects with the I/O Address management list 1310and when intactan entry is added to this list 1310 and a utilization permission is given.

[0108]Since the I/O Address range which the device which the second OS manages requires is stored in the kernel configuration information table 700 like the interrupt numberif it is referred tobefore it can know a requested address and the first OS will start device managementit can reserve an I/O Address.

[0109]By processing of Step 1202the thing which became independent of the first OS thoroughly and for which the second space only for OS is built becomes possible. In the device with which the second operating system manages the user program which operates on first OSand this exampleaccess to the input/output devices 116 and 117 becomes impossible by processing of Step 1204. It can forbid introducing the device driver using the interrupt number and I/O Address of the devices 116 and 117.

[0110]It becomes possible to introduce the second OS into the portion in which the first OS does not have a concern as an effect of processing of these two steps.

[0111]Continuing Step 1205 thru/or Step 1207 are the same as the initialization processing of the usual operating system. In initialization of the system device of Step 1205a kernel initializes the system device managed directly. A system device is a device assumed that are indispensable to execution of the first OSsuch as clock interruptionand the first OS always exists.

[0112]In Step 1206each initialization entry is performed about the object file which the kernel loader loaded. The initialization entry address is stored in the header part of an object file. Finallyan initial process is created (Step 1207).

[0113]Nextthe load procedure of the second OS in an embodiment of the

invention is explained. Drawing 14 is a flow chart which shows the load procedure of the second OS.

[0114]Firstit is necessary to read the object file of the second OS into the physical memory field assigned to second OS. Howeversince the physical memory field of the second OS cannot be written in from the first OS if it remains as it isit maps the assigned physical memory field temporarily in the virtual space of the first OS (Step 1401).

[0115]In Step 1402the object file of the second OS is read into the mapped field using the file reading procedure of the first OS. The form of the object file of the second OS presupposes that it is the same form as the object file form 800 of the first OS.

[0116]Nextthe page table for the second OS is created (Step 1403). This page table is also created in the field for the second OS. About the portion shared with the first OS at this timea page table is built so that it can be referred to also from the space of the second OS.

[0117]Let the field which loaded the device driver (henceforthsupport driver) which carries out interruption processing and management of common data about the common area 203 be the common area 203. The address with which this device driver was loaded can be known from the load object list 1104. The external reference of the kernel of the second OS is solved at continuing Step 1404. Howeverreference of other object files which can carry out the direct reference of the second OS is only public presentation reference of the function arranged in the common area 203and datai.e.a support driver. Thereforewith reference to the public presentation reference table 811 stored in the header part of the object file of a support driverthe external address 904 of the external reference tables 810 of the kernel object file of the second OS is determined here.

[0118]Nextthe address of public presentation reference of the second OS is written in the external reference address table to which it was assigned by the data area of the common area. Since the support driver used as a common area is read as a device driver of the first OS according to the mechanism of the first

OSit cannot be linked with public presentation reference of the second OS.
[0119]Herethe table which stores an external reference name required in the data area of a support driver and the external address corresponding to it is prepared beforehand. The execution code of a support driver describes that the call of the open function of the kernel of the second OS and reference of released data are carried out with reference to this table. And suppose that the address of public presentation reference of a support driver is written in the external address column of this table at the time of loading of the second OS.
[0120]Mapping of the physical memory field for the second OS which ended setting out of the second OS area and carried out the map to the kernel field of the first OS now is canceled (Step 1406).
[0121]Nextthe second context and OS discernment variable 1530 of OS of OS context table 1510 are set up (Step 1407). OS context is a data structure referred to when changing an execution operating systemand consists of a page table address value and an initial value of a stack pointer. Herethe initial address of the kernel stack of the second OS is set up for the address of the page table which carries out the map of the second OS as a page table register value as a stack pointer value. The value which shows that the first OS is performing is stored in OS discernment variable 1530. OS context table 1510 and OS discernment variable 1530 are mentioned later.
[0122]Nextthe initialization module of the second OS is performed (Step 1408). The change of operating system space follows on this. Another flow chart explains the change of an operating system. The initialization module of the second OS is public presentation referenceand the support driver can know the address.
[0123]Finally the address of the interrupt handler of the first OS registered into the present interruption table 104 at Step 1409It copies to the column 1522 of the hair drier of the interruption identification table 1520and an interruption table register value is changed into the address of the interruption table assigned to the support driver. This is carried out by change of the interruption table register

104 of the processor 101.

[0124]Whichever the operating system is performing at the time of interruption generatingan interruption table is changed into the table in a support driver because it is always necessary to interrupt the virtual address space of the processor 101 and the table needs to exist. The interrupt handler registered into an interruption table is also arranged in a support driver. Since it maps also in the virtual space of the second OS and is considered as the common area 203 at Step 1403the field of a support driver can be referred to at any time. The interruption processing of a support driver is mentioned later.

[0125]The interruption management information of the first OS is also changed in Step 1409. Although the data structure relevant to an interrupt inhibit level is changedspecificallythis is mentioned later.

[0126]The data structure stored in the common area 203 is explained. Drawing 15 is a figure showing the data structure stored in the data area 1500 of the common areas 203. According to drawing 15it explains in order.

[0127]1510 is OS context table. OS context table 1510 holds data required for the change between the first OS and the second OS. According to this embodimentthe first OS presupposes that it can run only when the second OS is an idle state. In this casewhat is necessary is just to return control to the first OSwhen the change to the second OS takes place at a certain time under first OS execution and execution of the second OS is completed.

[0128]Thereforethe number of the contexts which must be saved by each may be one. About the context of the first OSif the page table register value 1511 and the stack pointer value 1512 in the time of OS change being required are savedcontrol can be returned to the first OS after the second end of OS execution.

[0129]When changing control to the second OS from the first OSthe second OS is not operating. Thereforethe value of immobilization of a page table address and a stack pointer may be sufficient as the context of the second OS. The second page table register value 1513 and stack pointer value 1514 of OS are

set up when the second OS is loaded (Step 1407).

[0130]1520 is an interruption identification table. As for the interruption identification table 1520the value 1521 which shows which operating system processes interruptionand the address 1522 of the interrupt handler are recorded for every interrupt number of external interruption. If external interruption occursthe interrupt handler in the common area 203 will capture interruptionwill determine which OS is made to process with reference to processing OS1521 of this interruption identification table 1520and will pass control to the address of the hair drier 1522. The details of interruption processing are mentioned later.

[0131]1530 is OS discernment variable which stores the value which shows the operating system under execution. This variable 1530 is set as the degree of OS change in OS change procedure which begins from Step 1601. In interruption processingan interruption processing procedure is determined with reference to this variable 1530.

[0132]1540 is a delay interruption state variable which shows whether the interrupt of the device which the first OS has managed during execution of the second OS occurred. This variable 1540 is recording of which interrupt number the interrupt occurred. OS change procedure inspects this variable 1540when execution of the second OS is completedand it determines whether to start interruption processing (Step 1608).

[0133]The change procedure of an operating system is explained. Drawing 16 is a flow chart in an embodiment of the invention which shows the change procedure of an operating system. This change procedure is called during execution of the first OSand changes the second OS.

[0134]The procedure shown in drawing 16 receives as an argument the address of the module of the second OS performed after changing to the second OSand the argument passed to the module. The address of the module of the second OS can be known if the external reference address table set up in the common area 203 is referred to.

[0135]Firsta present stack pointer value and page table register value are saved

as a context of the first OS of OS context table 1510 at Step 1601 to begin. In Step 1601the present stack pointer value is saved 1512 and the value of the present page table register 105 is saved 1511.

[0136]It is not necessary to save on OS context table 1510 about other register contexts. What is necessary is just to save if necessary at the stack of the first OS.

[0137]After saving a stack pointer and a page table register valuethe address of the page table which carries out the map of the second OS to virtual space is set as the page table register 105 at Step 1602. This is recorded on 1513 of OS context table 1510. A stack pointer is set to second OS. This is also stored in the stack pointer 1514 of the second OS of the table 1600.

[0138]At the following step 1603the delay interruption state variable 1540 which shows the interruption state of the first OS is cleared. The state variable 1540 is a variable which records the generation state of interruption from the device by which it was generated during second OS executionand which the first OS has managed. This is cleared before performing the second OS.

[0139]And OS discernment variable 1530 which OS under present execution shows is rewritten to the value which shows the second OS (Step 1604). Since a stack pointerthe page table register 105and OS discernment variable 1530 must have a value which was always consistentwhere all the external interruption is forbiddenthey must perform the steps 1601 thru/or 1604 so far.

[0140]At continuing Step 1605control is moved to the address of the module passed as an argumentand control is passed to the second operating system. In this embodiment of this inventionthe first OS presupposes that it can perform only when the second OS is an idle statewhile the second OS is not performing that is. Thereforewhen processing of the second OS is completedcontrol returns to Step 1606.

[0141]In Step 1606each of the page table register value 1511 saved on OS context table 15100 at Step 1601 and the stack pointer value 1512 is recovered. It changes into the value which shows that the first OS is performing OS

discernment variable 1530 at continuing Step 1607. Processing of these two steps must also be performed where interruption is forbidden.

[0142]Nextexternal interruption of the device by which it was generated during execution of the second OS and which the first OS manages is processed. Firstin Step 1608the delay interruption state variable 1540 is inspected and it is inspected whether the interrupt occurred or not. When not having generatedit ends and OS change procedure returns to a caller.

[0143]Step 1609 is performedwhen that is not right and the interrupt has occurred. In this stepit records that the adjournment interruption state variable with which the first OS has managed interruption generated during execution of the second OS has unsettled interruption. Thenthe interruption processing of the first OS is started (Step 1610). When all the interruption processings are completedit returns to the caller of OS change procedure.

[0144]The interruption processing in an embodiment of the invention is explained. Drawing 17 is a flow chart which shows the interruption processing procedure of this embodiment. The module which performs this procedure is registered into the interruption table 107 of a processor as an interrupt handler. This interrupt handler is arranged to the common area 203 which can be referred to from both operating systems.

[0145]If an external interrupt occurs and an interrupt handler is started by the processor 101an interrupt handler will inspect an interruption factor and the device which generated interruption will judge it in the device which the first OS managesand the device which the second OS manages (Step 1701). This judgment is carried out from making an interrupt number into an index for the interruption identification table 1520and referring to the OS column 1521. When it is a device of the first OSit progresses to Step 1702and in the case of the device of the second OSit progresses to Step 1705. For exampleif it says by drawing 15 and an interrupt number will be 1it is interruption of the first OSand if it is the interrupt number 4it will become interruption of the second OS.

[0146]Step 1702 is performed when interruption is interruption of the device of

the first OS. In Step 1702OS which was being performed at the time of interruption generating is judged. This judgment is carried out with reference to OS discernment variable 1530. When OS under execution is the first OSit progresses to Step 1703and in the case of the second OSit progresses to Step 1704.

[0147]The processing which begins from Step 1703 is processing when the device which the first OS has managed generates interruption while performing the first OS. In Step 1703the processing which begins from Step 1701 does not existbut the interrupt handler of the first OS sets up a context seem to have received control from the direct processor 101. A context shows the contents of the stackand the contents of the register here. And control is passed to the interrupt handler of the first OS. The address of the interrupt handler of the first OS is stored in the hair drier column 1522 of the interruption identification table 1520. For exampleif it is interruption of the interrupt number 1it will sink below 1 as an index and will ask for a hair drier address with reference to an identification table.

[0148]In this casecontrol does not return to the procedure which begins from Step 1701but the first OS continues processing.

[0149]Step 1704 is performedwhen the device which the first OS has managed generates interruption while performing the second OS. In Step 1704the interrupt number of the device which generated interruption is recorded on the delay interruption state variable 1540. Processing of an interrupt handler is ended now. Processing of interruption in this case is performed when execution OS changes to the first OS (Step 1608).

[0150]When the generated external interruption is interruption of the device which the second OS managesit progresses to Step 1705 and inspects which OS is performing. HereOS under execution is judged by OS discernment variable 1530. When the first OS is performingand the second OS is performing to Step 1706it progresses to Step 1711.

[0151]When the interrupt of the device which the second OS manages occurs

during execution of the second OSit performs Step 1711. Step 1711 starts the interrupt handler of the second OS. The address of the interrupt handler of the second OS is recorded on the hair drier column 1522 of the interruption identification table 1520. If interrupt handler processing of the second OS is completed and control returnsit will endand this interrupt handler will also recover a context when it interruptsand will return control.

[0152]When external interruption of the device which the second OS manages occurs during execution of the first OSit performs Step 1706. In this caseprocessing of the second OS is given priority to and performed rather than execution of the first OS.

[0153]Firsta context is saved in Step 1706. A context here shows the contents of the stack required to recover a state when it interruptsand the contents of the registerwhen returning to the first OS after interruption processing is completed. This context is saved at the stack of the kernel of the first OS.

[0154]Thenchange of execution OS and starting of the interruption processing of the second OS are performed (Steps 1707 and 1708). This is performed by the procedure which begins from Step 1601.

[0155]When processing of the second OS is completedthe change to the first OS is performed (Step 1709)the context at the time of interruption is recovered (Step 1710)and processing of the first OS is resumed. Processing of Step 1709 does not necessarily need to be performed within the same module as the processing which begins from Step 1701. Processing returns to this module by the change to the first OS.

[0156]Processing of clock interruption currently shared between two operating systems is explained. Clock interruption is captured by the interrupt handler in ** and a common area. In this interrupt handlerthe interrupt handler for clock interruption of the second OS is performed first. The interrupt handler of the second OS is stored in the hair drier 2 column 1523. If execution of the interrupt handler of the second OS is completedinterruption processing of the first OS will be performed by processing which begins from Step 1702 of drawing 17. The

address of the first interrupt handler is stored in the hair drier column 1522.

[0157]Nextthe interrupt control portion of the first OS is explained. This is processing for interruption of the device which the second OS manages accidentally by the interrupt control of the first OS not to be carried out to prohibition.

[0158]The first OS assumes that interruption is controlled by the interrupt inhibit level. An interrupt inhibit level is a mechanism which is needed in order to realize exclusive control between the portion which operates by extension of the interruption processing in the kernel of an operating systemand the portion which is not so.

[0159]The first OS realizes an interrupt inhibit level by programming the interrupt control device 112. That isthe interruption mask register 502 of the interrupt control device 112 is programmedand external interruption is selectively used as a mask. Since the first OS is not known at all about the second OSwhen the first OS changes an interrupt inhibit levelthe mask of the interruption of the device of the second OS may be carried out. In order to prevent thisthe interrupt control portion of the first OS is changed.

[0160]Drawing 18 shows the data structure which the first OS that realizes an interrupt inhibit level has managed. 1800 is an interrupt inhibit level table. It is shown whether each interrupt level is expressed numerically and carries out the mask of the external interruption of what No. about each interrupt inhibit level. The place where the check of the table 1800 is attached shows the thing which carry out the mask of the interruption and to set up. For examplein the interrupt inhibit table 1800the interrupt inhibit level 0 shows that the mask of no interruption is carried out. The interrupt inhibit level 3 shows carrying out the mask of the interruption of the interrupt numbers 3 thru/or 5 with the interrupt control device 112. In the interrupt inhibit level 5the mask of all the interruption is carried out by the interrupt control device 112.

[0161]In this inventionthis interrupt inhibit level table 1800 is changed at the time of initialization of the second OS (Step 1409). In Step 1409about interruption

which the device which the second OS manages generatesan interrupt inhibit level table is changed so that the first OS may not carry out the mask of those interruption. Specifically with reference to the OS column 1521 of the interruption identification table 1520the check of the interrupt inhibit level table 1800 is cleared about the interrupt number which the second OS manages.

[0162]In this examplethe interrupt numbers 4 and 5 serve as interruption which the second OS processes. Thereforethe column of the interrupt numbers 4 and 5 of all the interrupt inhibit levels of the interrupt inhibit level table 1800 (all of 1801 and 1802) is cleared.

[0163]Therebyeven if the first OS changes an interrupt inhibit levelthe mask of the interruption of the device which the second OS manages will not be carried out.

[0164]The above enables it to operate two operating systems simultaneously by one computer.

[0165]When according to this invention adding change to the first OS and operating two operating systems simultaneouslySince a changed part is limited to the initialization portionthe device resource reservationand the interrupt inhibit control section of an operating system kernelit becomes possible to operate two operating systems simply.

[0166]In the method by a virtual machinein order to virtualize a physical memory and an I/O channelthe emulation of a privilege instruction is neededbut when software realizes thisan overhead becomes large and is a problem. In the virtual-machine methodit has hardware special for this overhead reduction more often. Howeverby determining beforehand the operating system which manages a device about each device in this inventionand determining further the range which can be used about a physical memory at the time of initializationAs operating systems did not interfere mutuallythey abolished control by the complicated software [ like ] in a virtual machineand they also made unnecessary hardware for improvement in the speed.

[0167]According to this inventionit is possible to add easily OS which

complements the function of the first OS. Also in conventional technologyit is possible to add a function new as a device driver to a kernel as a component of the first OSfor example. Howeverif kept as a component of the first OSthe component has a problem which can operate only under management of the first OS. That iswhen the first OS has stopped according to the obstaclethe added functional module cannot operateeither.

[0168]According to this inventioneven if the component which realizes a new function could be independently constituted with the first OS and the first OS has stoppedit becomes possible to continue only the functional module and to operate. This embodiment is mentioned later. If the functional module of which reliability is required is incorporated as the second OSeven when the first OS will have stoppedit becomes possible to realize a certain recovery. Thusthis invention is an invention which can serve as a means to realize high reliability-ization of a computer system.

[0169]By this embodimentit explained the second OS having had priority over the first OSand having performed processing. As for the first OSthe second OS's being able to operate only at the time of an idol and interruption of the second OS are giving priority to the second OS by processing immediately always. If OS which is fit for real time processing as the second OS is introduced by this even if the first OS does not conform to real time processingit will become possible to build the computer system excellent in the real-time-processing performancewith the feature of the first OS harnessed. For examplealthough it has GUI (Graphical UserInterface) excellent in the first OSwhen real-time-processing performance is missingConstruction of the computer system which was excellent also in GUI and was excellent also in real time processing by introducing the for [ real time processing ] operating system which has priority over the first OS and operates as the second OS can be performed.

[0170]Thusthis invention is the method of introducing without support of special hardware easily the function in which the first OS is missingand the function makes it possible to make it completely operate independently with the first OS

further.

[0171]Nexta second embodiment of this invention is described. A second embodiment is extension of the embodiment described so far. According to this embodimentintroduction of the second OS that continues operating even if the first OS stops according to an obstacle is realizable.

[0172]In addition to a first embodimentthe first OS run state variable 1550 is put on a common area. This variable 1550 stores the value which shows whether the first OS is carrying out normal operation or that is not right. This variable 1550 is processing when the second OS is loadedand is initialized to the value which shows normal operation.

[0173]Drawing 19 is a flow chart which shows the stop processing procedure of the first OS of a second embodiment of this invention. This procedure changes and mounts the module which performs stop processing of the first OS.

[0174]Firstif control comes to the stop processing module of the first OSit will be set as the value which shows that the first OS has stopped the first OS run state variable 1550 (Step 1901). Thenstop processing of the first OS is performed (Step 1902). The mask of the interruption to the first OS is carried out to the lastand interruption of the device which the second OS processes is permitted (Step 1903)and it waits until an interrupt occurs (Step 1904). If an interrupt occursexecution OS will be changed and the second OS will perform processing.

[0175]An execution OS change procedure is changed. According to a first embodimentthe change of execution OS was performed by the procedure which begins from Step 1601. According to a second embodimentafter performing the module of the second OS in this procedure that isthe first OS run state variable 1550 is inspected after Step 1605. Hereif the first OS run state variable 1550 has a value which shows that the first OS has stoppedwaiting for interruption will be performedwithout performing processing after Step 1606.

[0176]The above data structure and procedure enable it to continue execution of the second OSeven if the first OS stops. Although it presupposed that the stop processing module of the first OS is changed in this embodimentthe module

performed in a stop processing process when the first OS stops by error is changedand the same effect is realizableeven if it detects and sinks below the stop of the first OS and carries out waiting.

[0177]A 3rd embodiment of this invention is described. According to the embodiment described so farthe concurrency of two OS'setc. have been realized by changing a kernel main part. According to a 3rd embodimentthe function of the aforementioned embodiment is realizedwithout changing a kernel main part.

[0178]With the operating system which supports various kinds of hardwaresprocessing of hardware dependence may be separated and may comprise a kernel main part as another object file. For examplethey are a case where the interrupt control device 112 changes with computersand a case where the composition of the bus 109 differs and I/O Address space changes with computers.

[0179]It is the figure in which a code and data for drawing 20 to absorb the difference in the hardware used as the base of such an operating systemi.e.an interrupt control devicea busetc. showed the situation of the kernel field in case a kernel main part is in the separated object file.

[0180]There are a module performed by the kernel mode of the processor 101 and a data structure which an operating system manages in the kernel field 2000. The kernel main part 2001 has the code and data which process hardware un-depending of memory managementprocess schedulinga file systemetc. Between the kernel main part 2001 and the hardware dependence part 2002the agreement about the module which must provide the hardware dependence part 2002and the module which the kernel main part 2001 provides is defined. If the hardware dependence part 2002 is united with this agreement and builtit will become possible to operate this operating system on various computers.

[0181]It separates into another object file and the map of the processing of hardware dependence in which this agreement was followed is carried out to the kernel main part to the separated field 2002. The kernel main part 2001 and the hardware dependence part 2002 can call a mutual open module with the same

external reference mechanism as the case of a first embodimentand function as one kernel seemingly.

[0182]In such a caseit is possible to acquire the same effect as a first embodiment and a second embodiment by change of the object file which processes the separated hardware dependencewithout changing the object file of a kernel main part.

[0183]Specifically in processing of the separated object fileit is required that assignment of a physical memory is possiblefor interrupt level management processing to be changedand to be able to make a request to print out files of I/O resources. It interrupts into this object file with the interrupt handler which begins from Step 1701the table 107 is arrangedand it registers with the interruption table register 104 of a processor. And it enables it to refer to it also from the second OS by making this separated object file into the common area 203. By the abovethe same effect as a first embodiment of this invention can be acquired.

[0184]If the hardware dependence object file is the regulation with the module performed when the first OS stopsand the module is changedthe stop of the first OS can be detected and the same effect as a second embodiment of this invention can be acquired.

[0185]In this embodimentit is not necessary to change a kernel main part. It becomes feasible easily rather than changing the kernel main part which the portion which must be changed can limit further by this.

[0186]Nexta 4th embodiment of this invention is described. In the embodiment described so farit was a support driver and object files which were arranged to the common area 203such as a hardware dependence object file. Howeverthe module which must be truly arranged to the common area 203and data are only parts for the interrupt handler which begins from the interruption table 107 and Step 1701OS change procedure which begins from Step 1601and the data structure which were shown in drawing 15. When it enables it to refer to it also from the second OS by making into the common area 203 the whole object file which processes a hardware dependence part like [ in the 3rd example ]

especiallya possibility that the second OS will access the data structure of the first OS accidentally becomes highand is a problem.

[0187]According to a 4th embodimentthe method of showing it as the second OS by making only the specific section of an object file into the common area 203 is provided. According to this embodimentthe compiler which generates an object file needs to have a function in which the section which arranges an instruction code and data can be specified on a program.

[0188]The usual object file has a text section which contains an instruction code as a sectionand a data section containing data. In additionthe section for the common area 203 is added with the function of a compiler. What is necessary is to determine the address range of a common area section with reference to the section data 809 stored in the header part of an object fileand just to build a page table so that only the portion may be shown as the second OS.

[0189]The case where the object file containing the module which carries out hardware dependence processing is changed is explained as an example. It is not necessary to show the portion relevant to initializationfor exampleassignment of a physical memorya request to print out files of I/O resourcesand change of an interrupt level custodial area as the second OS among change parts. Only the amount of [ the interrupt handler which begins from the interruption table 107 and Step 1701OS change procedure which begins from Step 1601and / which were shown in drawing 15 ] data structure must be able to refer to it also from the second OS. It describes that a program arranges these into a common area sectionand the function of a compiler generates a common area section.

[0190]Drawing 21 shows the composition of the generated object file. 2100 shows the generated object file. 2101 thru/or 2104 of the header unit of the object file 2100 have described the data of the section included in the object file 2100. Among these2103 and 2104 are section data expressing the section newly created in the common areas 203. Corresponding sections are 2107 and 2108. If the page table of the second OS is constituted so that it may ask for the address of the sections 2107 and 2108 according to the contents of the section data 2103

and 2104 and the map only of those fields may be carried out to the kernel field of the second OSOther portions of the hardware dependence object file 2100 can be hidden from the second OS.

[0191]According to a 4th embodimentrather than the embodiment described until nowthe independency between OS's can be increased further and construction of a safe computer system with little interference between OS's is attained.

[0192]Nexta 5th embodiment of this invention is described. According to a 5th embodimentintroduction becomes possible about the second OS by the computer of a multiprocessor configuration.

[0193]Drawing 22 is a figure showing the computer system in a 5th embodiment of this invention. 2200 is a computer system. The computer 2200 has the two processors 2201 and 2202 and the main memory unit 2203. It has the memory storage 2204 which stores the computer boot program like a first embodiment.

[0194]About the processors 2201 and 2202the physical addresses which pass control shall differ in the time of starting a processorand the time of receiving interruption for initialization.

[0195]The initialization interruption processing program stored in the memory storage 2204 passes control to the address by making into a physical address the value stored in the physical address defined beforehand.

[0196]The device of the magnetic disk drive 2206the clock interruption generating device 2207and input/output device 2207 grade has connected via the bus 2209. It connected with the interrupt control device 2205and the device which generates interruption is further connected to the processors 2201 and 2202 via the interruption bus 2211. Each processor presupposes that interruption can be sent to other processors.

[0197]The interrupt control device 2205 is explained. The interrupt control device 2205 has a function for a multiprocessor configuration. In addition to the interrupt mask function of the interrupt control device 112 in a first embodimentthe interrupt control device 2205 has the function to specify to which processor or a processor group interruption from each device is notified.

[0198]Drawing 23 is a figure showing the composition of the interrupt control device 2205. It interrupts with the selecting arrangement 2301 and work of the mask register 2302 is the same as a 1st embodiment. In addition to themthe interrupt control device 2205 has interruption Redirection table 2310 and the interruption sending set 2305.

[0199]Interruption Redirection table 2310 is recording the interrupt number 2312 when notifying which processor or the value 2311 which shows whether interruption is notified to a processor group about each device connected to the interrupt control device 2205. Interruption Redirection table 2302 can be changed by an I/O instructionand can be set up freely.

[0200]It is set up in the example of drawing 23 deliver the interruption 0 and 1 to CPU0 and deliver the interruption 2 to CPU1.

[0201]In response to the signal from the selecting arrangement 2301the interruption sending set 2305 interrupts with reference to interruption Redirection table 2310and determines a report destination and an interrupt number. And it sinks below the signal showing a report destination and an interrupt numberand transmits to the bus 2211.

[0202]If the computer 2200 is startedit is constituted so that only the processor 2201 may start operationand the processor 2201 executes the boot program stored in the memory storage 2204. A boot program reads into the main memory 2203 the kernel loader stored in the magnetic disk drive 2206 like the case of a first embodimentand performs it. A kernel loader creates the parameter table 1100. According to a 5th embodimentthe data in which it is shown how many processors the computer 2200 has in the device list is added.

[0203]Initialization processing of the first OS is performed after loading of the first OS. It stores in the physical address which defined beforehand the address of the initialization routine for processors other than a non-booting processor in process of initializationand initialization interruption is sent to the processor 2202. If initialization interruption is receivedthe processor 2202 will execute the program stored in the memory storage 2204and control will cross it to non-

booting processor initialization routine. Non-booting processor initialization routine sets up a page table register and an interruption table registershifts to a virtual address modeand continues initialization processing.

[0204]According to a 5th embodiment of this inventionat the time of a request to print out files of the device for the second OS of Step 1204 of drawing 12a processor is also reserved as it is only second for OS. Hereit explains reserving the processor 2202.

[0205]In the case of a multiprocessor configurationinitialization interruption is sent to a non-booting processor by initialization of the system device of the initialization procedure of the first OS that begins from Step 1201. In this caseinitialization interruption will be sent to the processor 2202 from the processor 2201. In this inventioninitialization interruption will not be sent about the processor reserved. Even if initialization of a kernel is carried out to whether it carried outthe processor 2202 has not operated yet.

[0206]Initialization of the interrupt control device 2205 is also carried out in initialization of the system device of Step 1205. In initialization of the interrupt control device 2205it interrupts so that interruption of the device which the second OS manages may be sent to the processor 2202 with reference to the constitution data 704 of the second OS of the kernel configuration information file 700and Redirection table 2310 is set up.

[0207]In the initialization procedure of the second OS that begins from Step 1401 of drawing 14initialization routine is set as the address of the initialization routine of the second OSand initialization interruption is sent to the processor 2202 at Step 1407. Therebythe second OS starts a run on the processor 2202.

[0208]Unlike the 1st thru/or a 4th embodimentall interruption of the device which the second OS manages is sent to the processor 2202 in which the second OS is operating by the interrupt control device 2205. It becomes unnecessary for this reasonto change execution OS. The first OS will operate by the processor 2201 and the second OS will operate by the processor 2202. Thereforethe interruption processing which begins from Step 1701 also becomes unnecessary.

[0209]The second OS sets an original interruption table as the interruption table register of the processor 2202and can have an original interrupt handler. It is not necessary to change the interruption table of the first OS. Howeverwhen the first OS changes the interruption mask register 2302 of the interrupt control device 2205it is necessary to add change so that the mask of the interruption from the device of the second OS may not be carried out.

[0210]In a 5th embodimentit is more possible in construction of a powerful computer system than in the embodiment of the 1st thru/or 4. In the embodiment of the 1st thru/or 4only while the second OS was acting as an idol of the first OSit was able to operatebut in a 5th embodimentthe first OS can always be operatedalthough the processor is takenand the second OS can also operate simultaneously.

[0211]According to a 5th embodimentthe field which must be shared between the first OS and second OS can be made small. According to the embodiment of the 1st thru/or 4the interruption tablethe data structure which accompanies an interrupt handler and interruption processingand OS change code had to be put on the common area 203. According to a 5th embodimentthese all become unnecessary and can make low a possibility that mutual OS will break accidentally [ OS / of a partner ].

[0212]Nexta 6th embodiment of this invention is described. It is a method with which a 6th embodiment carries out the concurrency of two or more two or more OS's to the embodiment described until now being a method which carries out the concurrency of the two OS's.

[0213]Although controlled by a first embodiment for the second OS to have priority over the first OSand to performan execution priority can be set up among two or more OS's in the method explained here.For exampleabout interruptionprocessing of interruption which OS of a priority lower than the priority of OS under execution manages is postponed. When the interrupt which OS of a priority higher than the priority of OS under execution manages occursexecution OS is changed immediately and interruption processing is started.

[0214]When OS under execution calls the module of OS with its high priorityexecution OS is changed immediately and a module call is carried out. When [ that ] reverse (i.e.when the processing by the side of OS with a low priority is needed)it controls to postpone the processing demanded until the OS acquired the execution right.

[0215]Drawing 24 is a figure showing the computer configuration of a 6th embodiment of this invention. Although the computer configuration is the same as an old examplesigns that two or more OS's are loaded to the main memory unit 102 are shown. Each operating system can be loaded in the same procedure as the procedure which loads the second OS shown in Step 1401.

[0216]Drawing 25 is a figure showing the relation of two or more OS's notionally. A first embodiment shows here that two or more OS's are operating on one processor in addition to first OS to having been two OS's.

[0217]It is shown that the map of the common area which is a part of first OS is carried out to the logic space 2022503and 2504 of other 2nd3rdand Nth OS'sand it can use in common from all the OS's. Mapping of the common area to the logic space of each OS is carried out by the procedure shown in Step 1401.

[0218]It is shown that each OS has an external instrument which each manages. It is shown that the second OS manages the apparatus 116 and 117the third OS manages the apparatus 2505 and 2506and the Nth OS manages the apparatus 2507. What is necessary is just to store the I/O Address range for controlling these apparatusand an interrupt number in the kernel configuration information file 700 shown in drawing 7. Although drawing 7 described that only the configuration information for the second OS was storedthe configuration information on the third and fourth OS is stored in others. In the initialization procedure from Step 1201it is forbidden that the resources of not only the second OS but all the OS's other than first OS should be reservedand the first OS should access to the device which OS's other than first OS manage.

[0219]In the procedure which begins from Step 1201the main memory for the second OS is secured at Step 1202. This is considered as the processing which

secures the main memory for two or more OS's other than first OS. Although Step 1204 is processing which reserves the device which the second OS managesit is considered as the processing which reserves the device resources in which OS's other than first OS manage this.

[0220]The initialization procedure of the first OS in this embodiment is shown in drawing 26. Steps 2602 and 2604 are equivalent to Steps 1202 and 1204. Step 2604 reserves the device resources of OS's other than the first OS with reference to the kernel configuration information file 700.

[0221]At this timeit interrupts simultaneously and processing OS1521 of the management table 1520 is set up. To the configuration information on each OSthe priority of OS is also described besides device resources.

[0222]Drawing 27 is a figure showing the data structure arranged to the common area 203. As compared with the data structure shown in drawing 15the interruption identification table 1520OS discernment variable 1530and the delay interruption state variable 1540 are the same data structures. OS context table 2710 has a data structure which extended 1510.

[0223]The table 2710 saves the data which is needed when changing execution of each OS. The page table preset value 2701 and the stack pointer setting out 2702 show the address of the page table set up when calling the module of other OS's while performing a certain OSand a stack pointer. The page table preservation value 2703 and the stack pointer preservation value 2704 save the page table value and stack pointer value of OS with a lower priority when the change to OS with a high priority is carried out.

[0224]The run state 2705 stores the value which shows whether it is under operation about each OSand whether it is among a processor limited. Whether it is under operation here shows whether OS is started. It is not necessarily shown that it is under execution at a certain moment. The starting processing of each OS sets up the run state 2705.

[0225]That it is among a processor limited shows that the OS is not an idle state. That issince OS with a high priority is performingit is shown that it is in a run

waiting state. About the inside of a processor limitedit may describe which processing is waiting concretely.

[0226]The priority 2706 stores the execution priority of each OS. A priority is Step 2605 of the initialization processing procedure of the first OSand is read and set up from the configuration file 700.

[0227]The change procedure of execution OS is explained that the second OS is performing. The execution priority of OS assumes that the priority is highly set up for the third direction from the second and the second from the first. Suppose that the run of the process on third OS was attained by processing of the second OS. In this casethe process starting module inside third OS is performedand the process on third OS is scheduled. Heresince the priority is higher than the second OSthe third OS changes execution OS immediatelyand the module of the third OS is performed. In a spawn processa present page table address and stack pointer valueIt stores in the page table preservation value 2703 of the second OSand stack pointer preservation 2704 valuethe third page table preset value 2701 and stack pointer preset value 2702 of OS are set as a page table register and a stack pointerand execution OS is changed.

[0228]When the second OS calls the module of the first OSsince the priority of the first OS is lower than the priority of the second OSa call is postponed. In this casethe call factor is recorded run state 2705and a module call is carried out when the first OS acquires an execution right (i.e.when the second and third OS's end processing).

[0229]Drawing 28 is a flow chart which shows the change procedure of execution OS. The great portion of processing from Step 2801 is the same as the processing from Step 1601 shown in drawing 16. Although not taken into consideration about the priority of OSthis procedure should just carry out processing from Step 2801when the priority of OS may actually be called [ during the priority of call place OSand execution / before calling this procedure ].

[0230]The processing which begins from Step 2801 is called with switch destination OSand receives a modular address as an argument. In Step 2801a

present page table address and stack pointer are saved at the page table preservation value 2703 and the stack pointer preservation value 2704 of OS under present execution of OS context table 2710. OS under present execution can be judged by OS discernment variable 1530.

[0231]In the following step 2802the page table preset value 2701 and the stack pointer preset value 2702 of switch destination OS are acquired from OS context table 2710and the spawn process of a page table and a stack is carried out.

[0232]A delay interruption state is cleared in Step 2803. Above OS which a priority is performingthe delay interruption state 1540 of the device which OS of less than a switch destination manages is cleared.

[0233]For examplesuppose that the module of the first OS to the third OS is called. In this caseat Step 2803the delay interruption state 1540 of the device which the first OS and second OS manage is cleared.

[0234]In continuing Step 2804the module which set up the run state 2705 of OS under present execution of OS context table 2710 into the processor limitedset OS discernment variable as switch destination OS at Step 2805and was passed as an argument at Step 2806 is called.

[0235]OS of a switch destination returns control to Step 2807when the processing which should be carried out is lost. In Step 2807OS in a processor limited with the highest priority is found with reference to the run state 2705 and the priority 2706 of each OS of OS context table.

[0236]Step 2808 recovers the context of OS selected at Step 2807 from the page table preservation value 2703 and the stack pointer preservation value 2704 of OS context table 2710.

[0237]At the following step 2809it is set as OS which chose OS discernment variable 1530.

[0238]When it is recorded that there is processing postponed by the run state 2705 of OS context table 2700the processing postponed is performed (Step 2810).

[0239]Interruption delayed by continuing processing is processed. Step 2812

inspects whether with reference to the delay interruption state 1540the interrupt which OS selected at Step 2807 should process has occurred. If processing OS1521 of the interruption identification table 1520 is referred toit understands of which interrupt number selected OS has managed interruption.

[0240]As a result of the inspection of Step 2812when it judges with the interrupt of a processor limited having occurredit progresses to Step 2813. In Step 2813the data structure of OS chosen so that OS which had it chosen that the interrupt which must be processed had occurred could be recognized is set up. Continuing Step 2814 calls the interrupt handler 1522 of interruption to process with reference to the interruption identification table 1520. According to the recovered contextexecution of selected OS is resumed after the end of processing of a hair drier.

[0241]If there is no processor-limited interruptionexecution of selected OS will be resumed according to the context recovered as it was.

[0242]Interruption processing is explained. Drawing 29 is a flow chart which shows an interruption processing procedure. The routine which performs the procedure shown in drawing 29 interrupts as an interrupt handler of the processor 103and is registered into the table 400. It is arranged in the common area which can be referred to from all the OS's.

[0243]Procedure is explained. Firstin Step 2901OS which processes interruption from an interruption factor is determined. Processing OS is recorded on processing OS1521 of the interruption identification table 1520and is determined with reference to this.

[0244]If OS and interruption processing OS at the time of interruption generating are the same (Step 2902)the interrupt handler 1522 registered into the interruption identification table 1520 will be performed (Step 2903). The processing interrupted at the time of the end of hair drier processing is resumed.

[0245]When OS and interruption processing OS at the time of interruption generating differ from each otherthe priority of two OS's is measured (Step 2904). When the priority of OS under execution is higherit progresses to Step 2905and

the delay interruption state variable 1540 is set up and the interrupted processing is resumed.

[0246]When higher than OS which the direction of the priority of interruption processing OS is performingit progresses to Step 2906 and interruption processing is started.

[0247]In Step 2906the interrupt handler address 1522 is acquired from the interruption identification table 1520. Thenthe context at the time of interruption generating is saved (Step 2907)and an interrupt handler is called (Step 2908). The call of an interrupt handler is carried out by the procedure which begins from Step 2801 of drawing 28.

[0248]Thenwhen control returns to interrupted OSthe context at the time of interruption generating is recovered (Step 2910)and the interrupted processing is resumed.

[0249]Nextthe treatment module arranged to the common area 203 is described. In the common area 203OS change module from Step 2801 and the interrupt handler which begins from 2901 are arranged. The module called from all the other OS's is also arranged to the common area 203. For examplethey are a module which sinks below the interrupt handler of each OS and is registered into the identification table 1520and a module which sets up the run state 2705 of OS context table 2710.

[0250]Processing of external interruption is attained by each OS's calling the register module of the common area 203 by each initialization processingand registering an interrupt handler. When processing of other OS's of a priority lower than the priority of OS under execution is attained by a certain processingit being among a processor limited or the value which shows the factor of a processor limited must be set as the run state 2705. It sets up by calling the run state setting-out module in the common area 203 also in this case.

[0251]According to this embodimentexecution becomes possible about two or more OS's besides the first OS at one processorand it becomes possible to combine OS which specialized in the special function. For examplethe function of

the first OS is suppliable combining OS excellent in real time natureand OS which complements the reliability of a computer. Since each OS is independentlyvarious OS's can be combinedand the expansion of the first OS according to a use becomes possible. The effect acquired by the embodiment described until now is not spoiledeither.

[0252]Since a priority can be set up between OS'ssetting out of setting up the priority of real-time OS most highly is also possible. It enables this to use effectively the function which is excellent in each OS.

[0253]A 7th embodiment of this invention is described. This embodiment is extension of a 6th embodiment. By a 7th embodimentwhen a certain OS of the inside which is operating carries out an obstacle stopthe control method which other OS's which remained can execution continue is shown.

[0254]The error-processing module of each OS must set [ that it is under stopand ] up the run state 2705 of OS context table 2710when suspending execution of OS. This is carried out by the call of the run state setting-out module arranged to the common area 203 of said embodiment.

[0255]Each OS has a notice hair drier of OS stop called when other OS's suspend executionand registers it into the data structure in the common area 203 like an interrupt handler.

[0256]Nextprocessing of a run state setting-out module is explained. Drawing 30 is the flow chart which showed processing of this module. It explains that the third OS stops.

[0257]In Step 3001the run state 2705 of OS context table 2710 is set up.

[0258]The set-up run state is inspected in Step 3002. If are set as OS execution stops and it will becomeit will progress to Step 3003. Otherwiseprocessing of a module is ended.

[0259]In Step 3003the notice hair drier of OS stop of OS under operation whose priority is higher than stopped OS is called. A call is called by OS change from Step 2801. About OS with a low priorityit records that it is the waiting for OS stop hair drier execution on the run state 2706. The notice hair drier of OS stop of

each OS shall be registered into the data area arranged in a common area at the time of initialization of each OS like an interrupt handler.

[0260]In an execution OS spawn processthe processing which inspects whether OS of a switch destination has suspended execution is required. It explains according to the flow chart of drawing 28.

[0261]Firstbefore beginning a spawn processa switch destination must inspect whether it is [ be / it ] under operation. A spawn process will be terminatedif the run state 2705 of OS context table 2710 is inspected and it is not [ be / it ] under operation before Step 2801.

[0262]In selection of the next execution OS in Step 2807OS which is not working from the retrieval object of OS in a processor limited is excepted.

[0263]If the run state 2705 of selected OS is inspected and the execution waiting of the notice hair drier of OS stop is recorded before resuming execution of selected OSthe notice hair drier of OS stop will be performed.

[0264]Nextinterruption processing is explained. It explains according to the flow chart of drawing 29. Processing OS inspects in in operation after the interruption processing OS determination of Step 2901. When it judges with it not being under operation hereit is made to make the processing to which it canceled and sank below interruption resume.

[0265]The above processing enables other remaining OS's to continue operationeven if some OS's stop according to an obstaclewhile two or more OS's are operating simultaneously on one processor in addition to first OS.

[0266]Since the obstacle stop of each OS is notified to other OS'seven when processing by cooperating between OS'sit is possible to carry out error processing by each OS based on the noticeand the reliability as the whole computer can be improved.

[0267]According to the embodiment described until nowthe same OS may be included even if each OS is a mutually different OS.

[0268]

[Effect of the Invention]This invention is the method of introducing without

support of special hardware easily the function in which the first OS is missingand the function makes it possible to make it completely operate independently with the first OS further.

[0269]According to this inventionsince the changed part to the first OS is limited to the initialization portionthe device resource reservationand the interrupt inhibit control section of an operating system kernelit becomes possible to operate two operating systems simultaneously simply.

[0270]About the overhead by which it is generated by the method by a virtual machinein this invention. By determining beforehand the operating system which manages a device about each deviceand determining further the range which can be used about a physical memory at the time of initializationAs operating systems did not interfere mutuallythey abolished control by the complicated software [ like ] in a virtual machinethey also reduced the overheads by a command emulationand also made unnecessary hardware for improvement in the speed.

[0271]Also in conventional technologyalthough it was possible to have added a function new as a component of the first OS to a kernelwhen the first OS had stoppedthere was a problem on which the added functional module cannot operateeither. If the component which realizes a new function according to this invention is independently constituted with the first OSeven if the first OS will have stoppedit becomes possible to continue only the functional module and to operate. If the functional module in which reliability is demanded is incorporated as the second OSeven when the first OS will have stoppedit becomes possible to realize a certain recovery.

[0272]As for the first OSthe second OS's being able to operate only at the time of an idol and the interruption of the second OS can give priority to the second OS by processing immediately always. If OS which is fit for real time processing as the second OS is introduced by this even if the first OS does not conform to real time processingit will become possible to build the computer system excellent in the real-time-processing performancewith the feature of the first OS harnessed.

[0273]When Carnell of the first OS is divided into the Carnell main part and the object file which performs processing of hardware dependenceEven if it does not change the Carnell main partit becomes possible to operate two operating systems simultaneously by changing the latter object file. Since the portion which must be changed further is limited by thisit becomes feasible easily rather than changing the Carnell main part.

[0274]The field which needs to be referred to in common by the first OS and second OS using it if a section can be freely provided into an object file by description of a program by what is confined in a special section. The independency between OS's can be increased and construction of a safe computer system with little interference between OS's is attained.

[0275]According to this inventionsince the changed part to the first OS is limited to the initialization portionthe device resource reservationand the interrupt inhibit control section of an operating system kernelit becomes possible to operate two or more operating systems simultaneously simply.

[0276]Even when operation execution of two or more operating systems is carried outthe effect taken above can be acquired.

[0277]Also in conventional technologyalthough it was possible to have added a function new as a component of the first OS to a kernelwhen the first OS had stoppedthere was a problem on which the added functional module cannot operateeither. If the component which realizes a new function according to this invention is independently constituted with the first OSeven if the first OS will have stoppedit becomes possible to continue only the functional module and to operate. If the functional module in which reliability is demanded is incorporated as the second and third OSeven when the first OS will have stoppedit becomes possible to realize a certain recovery.

[0278]It becomes possible to combine OS which specialized in the special function by the ability of two or more OS's to be performed by one processor besides the first OS. For examplethe function of the first OS is suppliable combining OS excellent in real time natureand OS which complements the

reliability of a computer. The expansion of the first OS according to a use becomes possiblewithout being able to combine various OS's and spoiling other effectssince each OS is independently.

[0279]Even if the first OS does not conform to real time processing by the ability to set up an execution priority among two or more OS'sIf the second OS suitable for real time processing is set as a top priorityit will become possible to carry out by the ability to build the computer system excellent in the real-time-processing performancewith the feature of OS's other than the first and second OS harnessed.

[0280]When the kernel of the first OS is divided into the kernel main part and the object file which performs processing of hardware dependenceEven if it does not change a kernel main partit becomes possible to operate two or more operating systems simultaneously by changing the latter object file. The field which needs to be referred to in common by all the OS's using it if a section can be freely provided into an object file by description of a program by what is confined in a special section. The independency between OS's can be increased and construction of a safe computer system with little interference between OS's is attained.

## DESCRIPTION OF DRAWINGS

[Brief Description of the Drawings]

[Drawing 1]It is a figure showing the computer configuration of an embodiment of the invention.

[Drawing 2]It is a figure showing the computer configuration of an embodiment of the invention.

[Drawing 3]It is a figure showing the composition of a page table of an embodiment of the invention.

[Drawing 4]It is a figure showing the composition of an interruption table of an

embodiment of the invention.

[Drawing 5]It is a figure showing the composition of an interrupt control device of an embodiment of the invention.

[Drawing 6]It is a flow chart which shows the boot procedure of a computer of an embodiment of the invention.

[Drawing 7]It is a figure showing the composition of the kernel configuration information file of the first OS of an embodiment of the invention.

[Drawing 8]It is a figure showing the composition of an object file of an embodiment of the invention.

[Drawing 9]It is a figure showing the composition of an object file of an embodiment of the invention.

[Drawing 10]It is a figure showing the composition of an object file of an embodiment of the invention.

[Drawing 11]It is a figure showing the data structure of a kernel starting parameter table of an embodiment of the invention.

[Drawing 12]It is a flow chart which shows the initialization procedure of the first OS of an embodiment of the invention.

[Drawing 13]It is a figure showing the data structure of the device management table of the first OS of an embodiment of the invention.

[Drawing 14]It is a flow chart which shows the activation procedure of the second OS of an embodiment of the invention.

[Drawing 15]It is a figure showing the data structure which the first OS and second OS of an embodiment of the invention share.

[Drawing 16]It is a flow chart which shows the change procedure of execution OS of an embodiment of the invention.

[Drawing 17]It is a flow chart which shows the interruption processing procedure of an embodiment of the invention.

[Drawing 18]It is a figure showing the data structure for interrupt mask processing of the first OS of an embodiment of the invention.

[Drawing 19]It is a flow chart which shows the obstacle stop processing of the

first OS of a 2nd embodiment of this invention.
[Drawing 20]It is a figure showing the composition of the kernel field of the first OS and the second OS of a 3rd embodiment of this invention.
[Drawing 21]It is a figure showing the composition of an object file of a 4th embodiment of this invention.
[Drawing 22]It is a figure showing the composition of a computer system of a 5th embodiment of this invention.
[Drawing 23]It is a figure showing the composition of an interrupt control device of a 5th embodiment of this invention.
[Drawing 24]It is a figure showing the computer configuration of a 6th embodiment of this invention.
[Drawing 25]It is a figure showing the computer configuration of a 6th embodiment of this invention.
[Drawing 26]It is a flow chart which shows the initialization procedure of the first OS of a 6th embodiment of this invention.
[Drawing 27]It is a figure showing the data structure which all the OS's of a 6th embodiment of this invention share.
[Drawing 28]It is a flow chart which shows the change procedure of execution OS of a 6th embodiment of this invention.
[Drawing 29]It is a flow chart which shows the interruption processing procedure of a 6th embodiment of this invention.
[Drawing 30]It is a flow chart which shows OS run state setup steps of a 6th embodiment of this invention.
[Explanation of agreement]
A computer and 101 for 100 a processor and 102 a main memory unit and 103 An arithmetic unit104 an interrupt register and 105 a page table register and 106 An address conversion device107 -- an interruption table and 108 -- as for a clock generation device and 112a bus and 110 are [ an external I/O device and 118 ] interruption buses an interrupt control deviceand 113 thru/or 117 an

interrupting signal line and 111 a page table and 109.